Learning Transition Models with Time-delayed Causal Relations

Junchi Liang and Abdeslam Boularias[1]

***Abstract*— This paper introduces an algorithm for discovering implicit and delayed causal relations between events observed by a robot at arbitrary times, with the objective of improving data-efficiency and interpretability of model-based reinforcement learning (RL) techniques. The proposed algorithm initially predicts observations with the Markov assumption, and incrementally introduces new hidden variables to explain and reduce the stochasticity of the observations. The hidden variables are memory units that keep track of pertinent past events. Such events are systematically identified by their information gains. The learned transition and reward models are then used for planning. Experiments on simulated and real robotic tasks show that this method significantly improves over current RL techniques.**

I. Introduction

The advent of deep learning made a deep impact on most areas of computing, and provided dramatically improved solutions to many real-world problems. The appeal of deep learning is due to its simplicity that requires minimal design efforts, combined with a capability to learn complex functions from data. It did not take very long for this thrust to reach and transform the area of reinforcement learning (RL). The seminal work of [1] has shown that a simple neural network (DQN) could be trained to play Atari video games at a human level, using only raw screen images as observations and keyboard inputs as actions. The work on DQN paved the way for several new techniques that can be categorized under the general umbrella of *end-to-end visual RL*. These techniques avoid the tedious process of designing features manually, and rely on convolutional layers to automatically extract features from sensory inputs.

Despite the remarkable progress made by deep RL agents in reaching human-level performance and beyond [1], they continue to lag behind humans in terms of data efficiency. Humans can immediately figure out the effects of their actions on objects displayed on a screen after a few trials, and build a model for reasoning and planning in order to improve their scores. Model-based RL algorithms arguably require less data than model-free ones [2]–[4]. But learning models that are sufficiently accurate for planning is still a challenging problem [5]. Inaccurate predictions generally result in sub-optimal policies.

The difficulty in learning accurate predictive models can be mostly attributed to the partial observability of the states. In robotics, for example, the Markov condition is seldom verified. Future states and rewards often depend on the entire history of actions and observations.

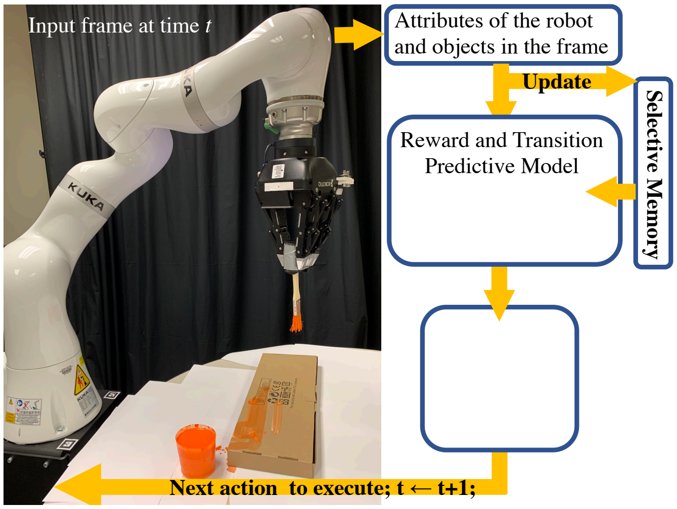

Input frame at time t
Attributes of the robot and objects in the frame
Update
Selective Memory
Reward and Transition Predictive Model
Next action to execute; t ← t+1;

Fig. 1. Overview of the proposed system and the robotic setup used in the experiment *Learning to Paint*

An example of that is unlocking a door and opening it afterwards. The first action changes the hidden variable that is the state of the door from `locked` to `unlocked`. Without a memory of past locking/unlocking actions, a robot cannot explain why the door opens sometimes and does not open at other times based only on an image of the door. The robot needs to infer the hidden *causal link* between the act of unlocking the door and the ability to open it later in the future. Other examples include filling or pouring liquids from containers, turning electric switches and unscrewing a lid and lifting it later in the future. In general, sequential object manipulation tasks involve changing states of objects in ways that cannot be easily perceived through vision, but their effects can be observed in the later stages of the task. Figure 1 shows an experiment performed in the present work, where a robot learns to paint. The causal link between dunking the paintbrush in a paint container and the appearance of paint on the surface of the box later when the brush is pressed against it is non-trivial because the robot performs a large number of random exploratory actions between the two events. The brush always looks the same, even after the paint in it has dried. Thus, it is important to remember the event of moving the brush into the container in order to predict future observations.

LSTM and GRU architectures are general-purpose tools for solving problems of partial observability by discovering and remembering pertinent information. They tend, however, to require large amounts of data, and they cannot be easily interpreted. To address these two issues, we present here an approach that combines the merits of general function

[1] The authors are with the Department of Computer Science of Rutgers University, Piscataway, New Jersey 08854, USA. {`jl2068`, `ab1544`}`@cs.rutgers.edu`

approximators such as neural networks with probabilistic graphical models for representing hidden variables. Given a stream of actions, observations and rewards, a neural network is trained to predict future observations and rewards. Simultaneously, a graphical model of causal relations between observations occurring at different time steps is also gradually constructed. The values of the variables in the graph are also provided to the neural network as additional inputs along with the observations. The learned predictive model is then utilized by the agent to select actions based on their predicted future rewards.

II. Related Works

Learning POMDPs: The Baum-Welch algorithm is an expectation-maximization technique that is traditionally used to learn hidden Markov models and POMDPs [6]. This algorithm requires knowing the number of hidden variables in advance. It is sensitive to the initial values of parameters and typically results in suboptimal solutions. Predictive state representations (PSRs) [7], [8] are an alternative model to represent partially observable environments without using hidden states. While parameters of PSRs can be learned with any consistent density estimator, discovering the core tests is still an open problem. Moreover, the learning complexity of these estimators is exponential in the length of history that needs to be stacked to predict future observations. A more efficient spectral approach proposed in [9] generalizes PSRs by including features of test outcomes and histories, instead of a stream of raw observations. For example, an indicative feature might be the number of times we saw a specific observation in the past three steps. The memory variables introduced in the present work are closely related to the indicative features. In contrast to [9], our algorithm reasons about the causal relations between different regions of the observation space and over different time intervals, and returns an explicit graphical representation of the discovered causal relations. Most recent efforts on learning partially observable dynamical models rely on recurrent neural networks (RNN) and LSTM techniques in particular [2]–[4], [10], [11]. While the success of LSTM is not fully understood, it is frequently attributed to the gating mechanisms that allows information to be retained for a long time, but also to be forgotten quickly.

Object-oriented RL: In vision-based RL, there is a clear physical structure that can readily be exploited. Images can be decomposed into segments of objects. Several models utilizing object-oriented representations for learning and planning have been proposed in the past [12]–[14]. Object-sensitive deep RL is a closely related idea proposed in [15]. More recent works focused on learning these models, such as the *interaction networks* [5], which can reason about how objects in complex systems interact. The *schema network*, proposed recently by [16], is an object-oriented generative physics simulator capable of disentangling multiple causes of events that occur in visual RL; it has been shown to increase transferability of skills within variations of Atari games. The schema networks are trained with an algorithm for structure learning in graphical models, but no hidden variables or delayed cause-effect relations were considered, which are the focus of our work. Object-oriented RL has not yet been applied to 3D games, to the best of our knowledge.

Memory models: Long-term dependencies in temporal models were considered in some recent works [17], [18]. For example, the method presented in [17] classifies trajectories based on the probabilities of future observations after an unknown number of steps. This approach is closely related to ours, except that we look into past events instead of future trajectories. [18] proposed another closely related approach based on using the reconstruction loss in recurrent neural nets as an auxiliary objective.

Attention mechanism: Attention mechanism is widely used for selecting specific features or specific parts of features dynamically according to the specified task. This mechanism is usually characterized by aggregating a collection of features through a weighted sum where weights are functions of inputs rather than learnable parameters. One of the successful applications of attention mechanisms is *caption generation for images*. A preeminent work, [19], defines attention weights as functions of visual features from different parts of an input image and from states of recurrent units in order to guide the architecture to focus on pertinent regions of the image as it generates a caption. The same mechanism was adopted in a more recent work [20]. Followup works explored various designs of the attention function. For instance, [21] investigates two new modules, one measures magnitudes of embedded features in attention computation while the other one takes object information as part of the input in order to obtain an attention module in a higher level. [22] takes advantage of attention mechanism to efficiently compress information from demonstrated trajectories in the setting of imitation learning. In this work, we focus on attention on specific events in the construction of memory units. Considering the forget and input gate as attention weights, LSTM inherently employs attention mechanisms [22]. We show that its attention tends to forget old events, while our proposed algorithm avoids this drawback.

III. Background and Notations

Formally, a Markov Decision Process (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ is a set of states and $\mathcal{A}$ is a set of actions. T is a transition function with $T(s'|a, s) = P(S_{t+1} = s'|S_t = s, A_t = a)$ for $s, s' \in \mathcal{S}, a \in \mathcal{A}$, and R is a reward function where $R(s, a, r)$ is the probability of receiving reward $r \in \mathbb{R}$ for executing a in s. A policy π is a distribution on the action to be executed in each state, defined as $\pi(s, a) = P(a_t = a|s_t = s)$. The value V^π of a policy π is the expected sum of rewards that will be received if π is followed, i.e., $V^\pi(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | S_0 = s, \pi, T, R]$.

In robotics, states generally cannot be fully observed. Instead, a robot perceives partial observations z_t, in the form of images, for example. The resulting process is a Partially Observable MDP (POMDP). Formally, a POMDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{Z}, T, F, R, \gamma)$ where $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ is an MDP, Z is a set of observations, and F is an observation likelihood

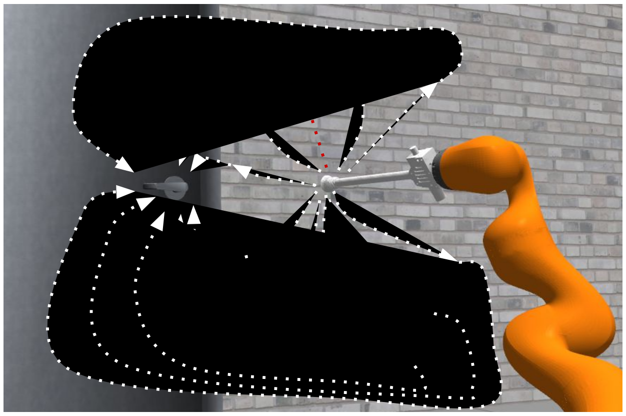


Fig. 2. Learning to open a door. The robot needs to first move its end-effector to the region surrounding the card reader to unlock the door, before attempting to open the door. The causal link between the two events is time-delayed and hidden in the middle of all possible random trajectories that are generated in the exploration phase. This work proposes an algorithm for learning such models.

function: $F(Z|S)$ is the probability of observing $Z \in \mathcal{Z}$ in state $S \in \mathcal{S}$.

We focus in this work on *object-oriented* POMDPs, where a state S is described by one or several visible attributes of objects, in addition to one or several hidden variables. In other terms, $S = (O^1, O^2, \dots, O^n, M^1, M^2, \dots, M^m) \in \prod_{i=1}^{n} \mathcal{O}_i \times \prod_{i=1}^{m} \mathcal{M}_i$, wherein O^i is an attribute of an object in the scene, and M^i is a hidden variable. The list of objects includes the robot or its end-effector. $\mathcal{O}_i$ and $\mathcal{M}_i$ denote the domains of visible variable O^i and hidden variable M^i, respectively, which can be discrete or continuous. For example, O^1 is the position of a specific object in the image, while O^2 is its velocity, O^3 is its size, O^4 is its numerical label, O^5 is a Boolean attribute that indicates if the object still exists in the scene or not, O^5 is the position of a second object in the image, and so forth. Observations correspond to object attributes, i.e., $Z_t = (O^i_t)_{i=1}^{n}$. To ease the notation, we also consider the reward signal R_t as one of the observable attributes O^i_t. Thus, the reward function is modeled and learned in the same way as the transition function. We focus hereafter on the general problem of predicting future states.

Hidden variables $M^1, M^2, \dots, M^m$ are unknown *a priori* and need to be inferred from the observable entities. Examples of hidden variables M^i include past events, or actions performed on the objects. In the previous example, a door can be unlocked or locked at a given time t, and the state of the door cannot be easily inferred from vision alone. The state of the door is then a hidden variable M^i_t of state S^t. Its existence can be inferred by discovering the causal link between the act of inserting and turning a key in a door lock and the subsequent ability to open the door later in the future, after executing several possibly unrelated actions. Transition function T is represented as a *Dynamic Bayesian Network* (DBN). We denote by $pa(X_t)$ the list of parents of variable X at time t. Thus, function T is defined as $T(S_{t+1}|S_t, A_t) = \Big(\prod_{i=1}^{n} P(O^i_{t+1}|pa(O^i_{t+1}), A_t)\Big)\Big(\prod_{i=1}^{m} P(M^i_{t+1}|pa(M^i_{t+1}), A_t)\Big)$, wherein the values of O^i_{t+1} and M^i_{t+1} are contained in S_t.

IV. Proposed Approach

The proposed algorithm receives a set of sampled data trajectories $\mathcal{D} = \{(z^l_0, a^l_0, \dots, z^l_h, a^l_h)\}_{l=1}^{L}$, collected by executing uniformly distributed random actions $\{a^l_t\}$ for exploration. The algorithm returns a set of memory variables $\{M^i\}_{i=1}^{m}$, and parents of each variable in $\{M^i\}_{i=1}^{m} \cup \{O^i\}_{i=1}^{n}$. We initially assume that there are no hidden variables, and incrementally create new ones only when needed to explain stochasticity of observations $\{z^l_t\}$. Algorithm 1 summarizes the main steps of this process.

A. Identifying highly stochastic variables

We start by setting k, the counter of hidden variables, to 0, and initialize the list of parents $pa(O^i_{t+1})$ for each object-attribute O^i with $\{O^1_t, \dots, O^n_t, A_t\}$. In other words, all attributes of objects in a given state are considered as relevant for predicting the next state. Next, factors $P(O^i_t|pa(O^i_t))$ of the transition function are estimated from sampled data trajectories $\mathcal{D} = \{(z^l_0, a^l_0, \dots, z^l_h, a^l_h)\}_{l=1}^{L}$. This can be achieved through any of the many existing density estimation techniques, such as frequency counts of discretized variables or the *Kernel Density Estimation* (KDE), as follows,

$$P(pa(O^i)) = \frac{\alpha w}{hL} \sum_{t=1}^{h} \sum_{l=1}^{L} \exp\Big(-w \sum_{X \in pa(O^i)} \|X - x^{i,l}_t\|_2 \Big),$$

$$P(O^i, pa(O^i)) = \frac{\alpha w}{hL} \sum_{t=1}^{h} \sum_{l=1}^{L} \exp\Big(-w \|O^i - o^{i,l}_t\|_2 - w \sum_{X \in pa(O^i)} \|X - x^{i,l}_t\|_2 \Big), \quad (1)$$

wherein $o^{i,l}_t$ (resp. $x^{i,l}_t$) is the observed value of variable O^i (resp. X^i) in trajectory l at time t. The algorithm maintains a list that contains state variables that need further explanation, i.e., variables with conditional probability distributions that have an entropy above a predefined threshold ϵ. For example, in the painting experiments, the variable that corresponds to the appearance of paint on the canvas is highly stochastic, even when conditioned on the pose of the brush with respect to the canvas and the executed motion. Conditional entropy is defined as,

$$\mathbf{H}(X|Y_1, \dots Y_m) = - \sum_{x \in \mathcal{X}} \sum_{y_1 \in \mathcal{Y}} \cdots \sum_{y_m \in \mathcal{Y}} \Big(p(x, y_1, \dots, y_m) \log \frac{p(x, y_1, \dots, y_m)}{p(y_1, \dots, y_m)} \Big), \quad (2)$$

The algorithm is safeguarded against entering infinite loops by upper bounding the number of parents per variable by max_var. The algorithm processes the variables, one at a time, until the open list is empty.

B. Searching for causes of uncertainty

The next step of the proposed algorithm consists in searching in the history of executed actions and observed object-attributes for events that can explain the stochasticity of each one of the identified highly stochastic variables, denoted by

X (line 9). To this end, the algorithm iterates through all attributes O^i of objects (lines 12-23). For each attribute O^i of each object present in the scene, including the robot's end-effector, we search in the space $\mathcal{O}^i$ of all possible values for the region that brings the maximum information gain to the distribution of variable X. The region is defined as hyper-ball, with a center $c_i \in \mathcal{O}^i$ and a radius $r_i \in \mathbb{R}$. For example, c_i could be a 6-dimensional point in space that corresponds to the pose of the robot's end-effector (or the manipulated object), and r_i is a tolerance threshold. Searching in the entire space $\mathcal{O}^i$ is computationally inefficient. Therefore, the proposed algorithm concentrates the search in the regions of space covered in the exploration data $\mathcal{D}$. We thus start by estimating the distribution of O^i as,

$$P(O^i) = \frac{\alpha w}{hL} \sum_{t=1}^{h} \sum_{l=1}^{L} \exp\Big(- w \|O^i - o_t^{i,l}\|_2 \Big). \qquad (3)$$

The center of the hyper-sphere is sampled from $P(O^i)$, while its radius r_i is sampled uniformly in the interval $[0, radius(\mathcal{O}^i)]$.

The center and radius of the sampled hyper-sphere are optimized in the next step (lines 16-19) by following the gradient of the information gain that occurs from adding the event of visiting the hyper-sphere into the list of parents of variable X. The information gain is defined as,

$$IG\big((X|pa(X)), Y\big) = \mathbf{H}(X|pa(X)) - \mathbf{H}(X|pa(X), Y).$$

Therefore, selecting a new parent Y that maximizes $IG\big((X|pa(X)), Y\big)$ corresponds to selecting a parent that minimizes $\mathbf{H}(X|pa(X), Y)$. In our case, Y corresponds to the event of visiting $\texttt{Ball}(c_i, r_i)$ at some point in the past. Thus, we first define $\hat{\mathbf{H}}$, a form of $\mathbf{H}$ that is parameterized by c_i and r_i, then compute the gradients of $\hat{\mathbf{H}}\big(X|pa(X), O^i \in \texttt{Ball}(c_i, r_i)\big)$ with respect to both c_i and r_i, and use them to update both parameters.

Boolean events $O^i \in \texttt{Ball}(c_i, r_i)$ are discrete and non-differentiable with respect to c_i, r_i. We thus employ a continuous relaxation where $P\big(X, pa(X), O^i \in \texttt{Ball}(c_i, r_i)\big)$ is defined as

$$\frac{\alpha w}{hL} \sum_{l=1}^{L} \sum_{t=1}^{h} \exp\Big(- w \|X - x_t^l\|_2 - w \sum_{Y \in pa(X)} \|Y - y_t^{i,l}\|_2 \Big)$$
$$\exp\Big(- w \min_{t' \in \{1,\dots,t-1\}} \|o_{t'}^i - c_i\|_2 - r_i \Big).$$

$\hat{\mathbf{H}}\big(X|pa(X), O^i \in \texttt{Ball}(c_i, r_i)\big)$ is then defined by using $P\big(X, pa(X), O^i \in \texttt{Ball}(c_i, r_i)\big)$ in Equation 2. Although the $\texttt{min}$ operator in this definition is not differentiable, the gradient of $\hat{\mathbf{H}}$ can still be computed analytically at any point (c_i, r_i) using well-known derivatives of exponential functions, except on small boundaries in the space $\mathcal{O}^i$, which are the boundaries of the *Voronoi* regions generated by the data points $\{o_t^i\}$.

C. Creating memory units

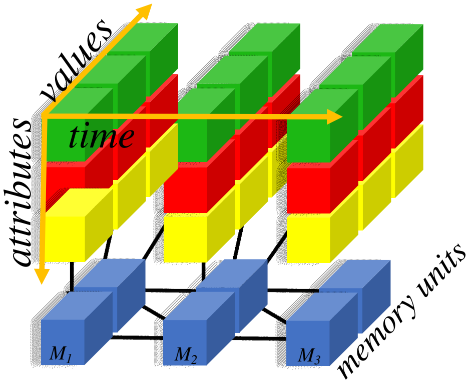


Fig. 3. Causality graph returned by Algorithm 1.

After identifying key event $O^i \in \texttt{Ball}(c_i, r_i)$ and optimizing (c_i, r_i) through gradient-descent, we create a binary memory variable $M^{(k)}$ associated with it. Therefore, $M^{(k)} \in \{0, 1\}$. Binary hidden variables $\{M^{(k)}\}$ are used to carry pertinent information over time and preserve the Markov assumption in the transition model. The transition function for a memory variable $M^{(k)}$ associated with object-attribute O^i is defined as follows.

$$P\big(M_0^{(k)} = 0\big) = 1,$$
$$pa(M_{t+1}^{(k)}) = \{M_t^{(k)}, O_t^i\},$$
$$P\big(M_{t+1}^{(k)} = 1 | pa(M_{t+1}^{(k)})\big) = 1 \text{ if } \big(M_t^{(k)} = 1 \vee \|o_t^i - c_i\|_2 \leq r_i\big),$$
$$P\big(M_{t+1}^{(k)} = 0 | pa(M_{t+1}^{(k)})\big) = 1 \text{ if } \big(M_t^{(k)} = 0 \wedge \|o_t^i - c_i\|_2 > r_i\big).$$

Therefore, memory unit M_{t+1}^i is deterministic. It is initialized with 0, and it preserves its value over time until the event $\|o_t^i - c_i\|_2 \leq r_i$ occurs. This event corresponds to object O^i being inside the sphere (c_i, r_i) at some time t.

This process is repeated until the entropy of variable X drops below a pre-defined threshold after identifying from the history data all hidden causes of X. Note also that the same object attribute O^i can be associated with more than one memory unit $M^{(k)}$. This happens for example when different regions, with different centers and radii, in space $\mathcal{O}^i$ of object O^i need to be visited in order for X to take a certain value. For example, we show in the next section how a robot can learn to identify four lug nuts that need to be loosened before it can remove a tire. The locations of the nuts are not known *a priori* to the robot.

V. Experiments

The proposed approach is evaluated on two tasks. These two are robotic experiments using the *Gazebo* simulator, we also deploy the proposed approach on a real robot for the painting task.

A video of the experiments is attached as a supplementary material and uploaded to https://bit.ly/2TeGlzm.

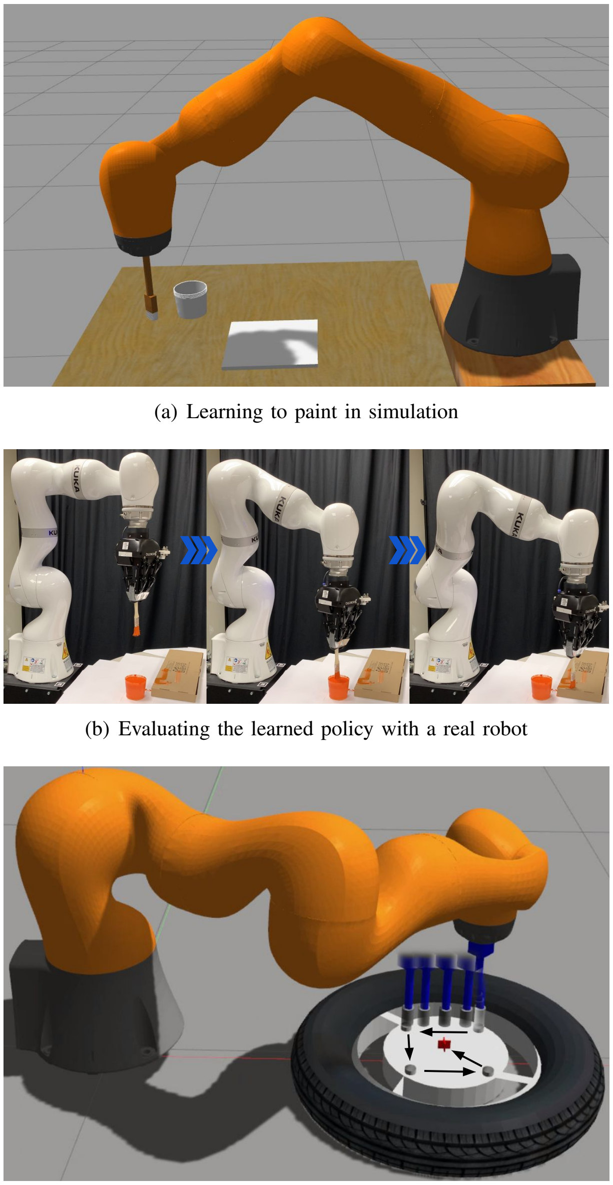
A. Robotic Experiments

1) Painting: We formulate the painting problem illustrated in Figure 4 as a POMDP, where the state space corresponds to the position of the paintbrush attached to the robot's end-effector, and a hidden binary variable that indicates if the paintbrush is loaded. It is important to note that the robot can observe only the position of the brush, and

Algorithm 1: Greedy Causal Graph Construction

```
Input: Set of observable variables $\{O^i\}_{i=1}^n$, and sampled data
       trajectories $\mathcal{D} = \{(z_0^l, a_0^l, \dots, z_h^l, a_h^l)\}_{l=1}^L$;
Output: Set of memory variables $\{M^i\}_{i=1}^m$, and parents of each
        variable in $\{M^i\}_{i=1}^m \cup \{O^i\}_{i=1}^n$ ;
$openList \leftarrow \emptyset$; $k \leftarrow 0$;
/* Identifying highly stochastic variables */
for $i := 1; i \leq n; i \leftarrow i + 1$ do
    $\forall t \geq 0 : pa(O_{t+1}^i) \leftarrow \{O_t^1, \dots, O_t^n, A_t\}$;
    Estimate $P(O^i, pa(O^i))$ from data $\mathcal{D}$ using Equation 1;
    Estimate $P(pa(O^i))$ from data $\mathcal{D}$ using Equation 1;
    Compute conditional entropy $\mathbf{H}(O^i|pa(O^i))$ from
      $P(pa(O^i))$ and $P(O^i, pa(O^i))$ by using Equation 2;
    if $\big(\mathbf{H}(O^i|pa(O^i)) > \epsilon\big)$ then $openList.push(O^i)$ ;
while $openList \neq \emptyset$ do
    $X \leftarrow openList.pop()$;
    /* Searching for causes of uncertainty */
    repeat
        $max\_gain \leftarrow 0$;
        foreach $i \in \{1, \dots, n\}$ do
            Sample an initial center-point $c_i \sim P(O^i)$;
            Sample an initial radius
              $r_i \sim Uniform\big(0, radius(\mathcal{O}^i)\big)$;
            repeat
                $\nabla_{c_i}\hat{\mathbf{H}} = \frac{\partial}{\partial c_i}\hat{\mathbf{H}}\big(X|pa(X), O^i \in$
                  $\mathtt{Ball}(c_i, r_i)\big)$;
                $c_i \leftarrow c_i - \alpha_{center}\nabla_{c_i}\hat{\mathbf{H}}$;
                $\nabla_{r_i}\hat{\mathbf{H}} = \frac{\partial}{\partial r_i}\hat{\mathbf{H}}\big(X|pa(X), O^i \in$
                  $\mathtt{Ball}(c_i, r_i)\big)$;
                $r_i \leftarrow r_i - \alpha_{radius}\nabla_{r_i}\hat{\mathbf{H}}$;
            until $\Big((\|\nabla_{c_i}\hat{\mathbf{H}}\|_2 < \epsilon_1) \wedge (\|\nabla_{r_i}\hat{\mathbf{H}}\|_2 < \epsilon_2)\Big)$;
            if $\hat{\mathbf{H}}\big(X|pa(X), O^i \in$
              $\mathtt{Ball}(c_i, r_i)\big) - \mathbf{H}(X|pa(X)) > max\_gain$ then
                $max\_gain \leftarrow \hat{\mathbf{H}}\big(X|pa(X), O^i \in$
                  $\mathtt{Ball}(c_i, r_i)\big) - \mathbf{H}(X|pa(X))$;
                $i_{max} \leftarrow i$;
        /* Adding hidden variables */
        Create a memory unit $M^{(k)}$ associated with the event
          $O^{i_{max}} \in \mathtt{Ball}(c_{i_{max}}, r_{i_{max}})$;
        $pa(X) \leftarrow pa(X) \cup \{M^{(k)}\}$;
        $k \leftarrow k + 1$;
    until $\big(\mathbf{H}(X|pa(X)) < \epsilon\big) \vee \big(|pa(X)| \geq max\_var\big)$;
```

not if it is loaded or unloaded. The action space corresponds to movements of the end-effector in six directions. We use the *Gazebo* simulator with the *MoveIt!* path planner to move the end-effector between two adjacent cells. The length of each episode is 100 time-steps. When the brush goes inside the paint bucket, the binary variable switches from *false* to *true* and remains so until the end of the episode. This simple transition is difficult to learn because the binary switch variable can never be observed, there is no immediate evidence related to it, and the robot is even unaware of its existence a priori. If the paintbrush was loaded, the robot receives a reward of +1 when the brush touches the canvas. In all other cases, the received reward is 0. The reward function is also unknown and needs to be learned from the observed trajectories. Given that the data is collected with a random policy, the time difference between dipping the brush in the bucket and touching the canvas can be arbitrarily long.



(a) Learning to paint in simulation

(b) Evaluating the learned policy with a real robot

(c) Learning to change a tire

Fig. 4. Tasks considered in the experiments

The robot receives data sequences $\{(z_t, a_t, R_t)\}_{t=1}^{100}$, where z_t is a 3D position of the brush, a_t is a moving direction, and R_t is an immediate reward. No other information is provided. The robot is then tasked with learning to paint.

Using the proposed algorithm, the robot learned a state transition model and a reward function from data. The learned model was then used by the value iteration algorithm to return a policy. The state transition function here is trivial, but the reward function is less trivial and involves a hidden variable related to the loading status of the brush. The robot first noticed that the received rewards are seemingly stochastic and searched for past events that might be behind the variance in the received rewards. Searching for such events is challenging because the data is simply a collection of Brownian-like motions of the end-effector gathered using a uniform exploratory policy. Nevertheless, the algorithm

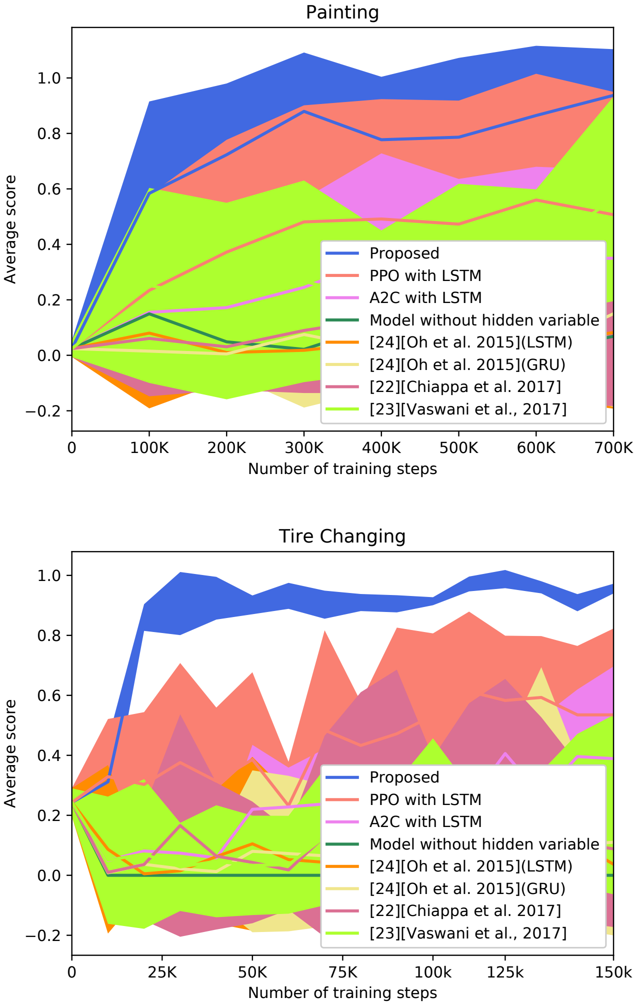


Fig. 5. Average reward per test episode as a function of the number of time-steps in the training data. *Stacking History* refers to a variant of our architecture where the hidden variables are replaced with a long history of actions, observations and rewards.

Task	Proposed	LSTM	GRU	[23]	[24]
Painting Reward 0	0.99/0.99	1/0.99	1/0.99	1/0.99	0.99/0.99
Painting Reward 1	0.90/0.96	0/0	0/0	0/0	0.08/0.152
Tire Reward 0	0.99/0.99	1/0.99	1/0.99	1/0.99	1/0.99
Tire Reward 1	0.96/0.97	0/0	0/0	0/0	0/0

TABLE I

RECALL / PRECISION IN PREDICTING THE TWO VALUES OF REWARDS. LSTM AND GRU ARE BASED ON [25].

succeeds in localizing a region of state space that must be visited by the brush before pressing it on the canvas in order to make paint appear on the canvas. The algorithm systematically creates a memory unit associated with the event of visiting that specific region of the state space, which corresponds to dipping the brush in the bucket.

Results reported in Figure 5 show that our algorithm converges to a nearly 100% success rate. The results are averaged over 200 test episodes and five different initial positions of the paint bucket and the canvas.

We also compared the proposed method to the state-of-the-art model-free RL algorithm Proximal Policy Optimization (PPO), with an LSTM unit. PPO is implemented with a neural network that receives current position as input. The input layer is followed by fully a connected layer with 128 and 256 units, followed by an LSTM layer with size of 256. The learning rate is 10^{-4}, batch size is 4, the number of epoch is 2, discount factor is set to 0.99 and the number of workers is 8. Despite performing significantly better than a version of our method that does not search for the hidden variables, PPO underperformed in this task. Another popular model-free RL algorithm A2C [26] is implemented with the same setting as PPO. Its learning curve shows a similar pattern as PPO but it increases more slowly. We also compared our approach to a recent model-based RL approach for POMDPs [25], where LSTM is used in one version and GRUs are used in another version. Both versions failed to learn to accurately predict the reward function using the same data used by our approach, as shown in Table I. The model learned by [25] was used for planning in a Monte Carlo tree search where the groudtruth optimal path was always provided intentionally within the set of sampled paths. But the optimal path was not selected by the model and the obtained policies were suboptimal, as shown in Figure 5. A more recent variant of [25], another model-based method [23] is evaluated. But it also does not manage to find the optimal path. Attention model [24] is designed to focus on parts of sequences, so we evaluate whether it can find the target region in this task. Applying the same planning procedure as above, its performance is better than the other two model-based baselines, but it is still worse than the proposed approach. So even though its performance may keep increasing as more data is available, it is clearly less efficient than the proposed approach.

2) Tire Removal: The painting experiments involve only one hidden variable. To test the proposed algorithm on problems with more variables, we designed a second task in Gazebo where the robot is tasked with removing a tire. We assume that the robot is already equipped with an automatic drill on its end-effector, and the task consists in placing the drill on the lug nuts to loosen them before moving to the center of the wheel to take it off. The problem is formulated in the same way as in the painting task, except that the hidden variables now correspond to the status of each of four lug nut (tight vs. loose). The wheel can be taken off only when the end-effector is placed at the center of the wheel, after placing it on four specific points corresponding to the lug nuts. A reward of 1 is given when the task is successfully finished, all other states have a reward of 0. Results are averaged over five different positions of the nut lugs.

Results in Figure 5 and Table I confirm that the proposed approach discovers the causal link between visiting four specific regions of the state space and receiving a positive reward at the end of the episode. As the number of hidden variables increases, baseline algorithms easily get trapped in local minima. The resulting learned policies tend to go to the wheel's center after only loosening one or two lug nuts, which causes their performance to drop. Because of the clipped loss, PPO agents have limited policy changes at each update. Although it cannot improve much from the initial random policy, PPO still avoids getting trapped in a local minimum policy. A2C learns a similar policy in the beginning, but gradually improves later. In all four model-based baselines, [24] outperforms others in the painting experiment but it cannot show a similar advantage in this second task. Simultaneously keeping track of multiple regions may not be trivial for current attention models.

VI. Conclusion

Model-based RL algorithms are known to be data-efficient alternatives to model-free ones, but also to be highly sensitive to modeling inaccuracies. Such inaccuracies are often due in robotics to the partially observable nature of states that makes predicted observations and rewards stochastic. In this work, we presented a new algorithm that searches for past events that can be the hidden causes behind the perceived noise in the future. The proposed model systematically builds a memory of these events and the times of their occurrences. The proposed approach searches for confounding hidden variables that once identified, can help reduce the noise of the predictions. Future works include deploying tire removal experiments on a real robot, as well as extending this framework to learning by imitation.

Acknowledgement

This work was supported by NSF awards 1734492, 1723869 and 1846043.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," 2018.

[3] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 512–519, 2016.

[4] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 2786–2793.

[5] P. W. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 4502–4510.

[6] L. R. Rabiner, "Readings in speech recognition," A. Waibel and K.-F. Lee, Eds. Morgan Kaufmann Publishers Inc., 1990, ch. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, pp. 267–296. [Online]. Available: http://dl.acm.org/citation.cfm?id=108235.108253

[7] S. Singh, M. R. James, and M. R. Rudary, "Predictive state representations: A new theory for modeling dynamical systems," in *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, ser. UAI '04. Arlington, Virginia, United States: AUAI Press, 2004, pp. 512–519. [Online]. Available: http://dl.acm.org/citation.cfm?id=1036843.1036905

[8] A. Boularias and B. Chaib-Draa, "Predictive representations for policy gradient in pomdps," in *ICML 2009*, Max-Planck-Gesellschaft. New York, NY, USA: ACM Press, Jun. 2009, pp. 65–72.

[9] B. Boots, S. M. Siddiqi, and G. J. Gordon, "Closing the learning-planning loop with predictive state representations," *Int. J. Rob. Res.*, vol. 30, no. 7, pp. 954–966, Jun. 2011.

[10] C. Downey, A. Hefny, B. Li, B. Boots, and G. J. Gordon., "Predictive state recurrent neural networks," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2017.

[11] K. Choromanski, C. Downey, and B. Boots, "Initialization matters: Orthogonal predictive state recurrent neural networks." in *Proceedings of the Sixth International Conference on Learning Representations (ICLR)*, 2018.

[12] C. Diuk, A. Cohen, and M. L. Littman, "An object-oriented representation for efficient reinforcement learning," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08, 2008, pp. 240–247.

[13] J. Scholz, M. Levihn, C. Isbell, and D. Wingate, "A physics-based model prior for object-oriented mdps," in *Proceedings of the 31st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, E. P. Xing and T. Jebara, Eds., vol. 32, no. 2, Bejing, China, 22–24 Jun 2014, pp. 1089–1097.

[14] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, "Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks," *CoRR*, vol. abs/1609.02993, 2016.

[15] Y. Li, K. Sycara, and R. Iyer, "Object-sensitive deep reinforcement learning," in *GCAI 2017. 3rd Global Conference on Artificial Intelligence*, ser. EPiC Series in Computing, C. Benzm\"uller, C. Lisetti, and M. Theobald, Eds., vol. 50. EasyChair, 2017, pp. 20–35.

[16] K. Kansky, T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, D. S. Phoenix, and D. George, "Schema networks: Zero-shot transfer with a generative causal model of intuitive physics," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 1809–1818.

[17] A. Neitz, G. Parascandolo, S. Bauer, and B. Schölkopf, "Adaptive skip intervals: Temporal abstraction for recurrent dynamical models," in *Advances in Neural Information Processing Systems*, 2018, pp. 9816–9826.

[18] T. H. Trinh, A. M. Dai, M.-T. Luong, and Q. V. Le, "Learning longer-term dependencies in rnns with auxiliary losses," *arXiv preprint arXiv:1803.00144*, 2018.

[19] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 2048–2057.

[20] W. Jiang, L. Ma, Y.-G. Jiang, W. Liu, and T. Zhang, "Recurrent fusion network for image captioning," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 499–515.

[21] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang, "Bottom-up and top-down attention for image captioning and visual question answering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6077–6086.

[22] Y. Duan, M. Andrychowicz, B. C. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, "One-shot imitation learning," in *NIPS*, 2017, pp. 1087–1098.

[23] S. Chiappa, S. Racaniere, D. Wierstra, and S. Mohamed, "Recurrent environment simulators," *arXiv preprint arXiv:1704.02254*, 2017.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[25] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. P. Singh, "Action-conditional video prediction using deep networks in atari games." in *NIPS*, 2015, pp. 2863–2871.

[26] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1928–1937.